NTATION PAGE

Form Approved
OMB No 0704-0188

ated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1991	Final Report 1988-91

4. TITLE AND SUBTITLE
Inference and Decision Mechanisms in Artificial Intelligence: Final Report

5. FUNDING NUMBERS
DAAL03-88-K-0082

6. AUTHOR(S)
D. W. Loveland (Principal Investigator)
A. W. Biermann (Co-Principal Investigator)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Science Department
Duke University
Durham, NC 27706

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARO 25795.11-MA-AI

11. SUPPLEMENTARY NOTES
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This is the final report for ARO Grant DAAL03-88-K-0082, with investigators D.W. Loveland, A.W. Biermann and G. Nadathur. This grant impacted several projects undertaken by these investigators within the Duke C.S. Department. The METEOR theorem proving project focussed on a parallel implementation of the Model Elimination proof procedure, but discovered that the sequential version is also very powerful. The Near-Horn Prolog project addresses disjunctive logic programming, which extends Horn clause logic (Prolog) by allowing clauses with multiple positive literals. The λProlog project investigates foundational and implementation related aspects of a Prolog extension that incorporates higher-order logic terms and new search primitives into the Horn clause logic framework. The resulting language has been shown to be useful for prototyping new inference-oriented software. The final project is really several projects in learning; foundational, utilizing connectionism, and learning real-time programs.

14. SUBJECT TERMS
Final report, inference, decision, logic programming, learning, theorem proving

15. NUMBER OF PAGES
15

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std 239-18
298-102

Inference and Decisian Mechanisms in Artificial Intelligence

Final Report

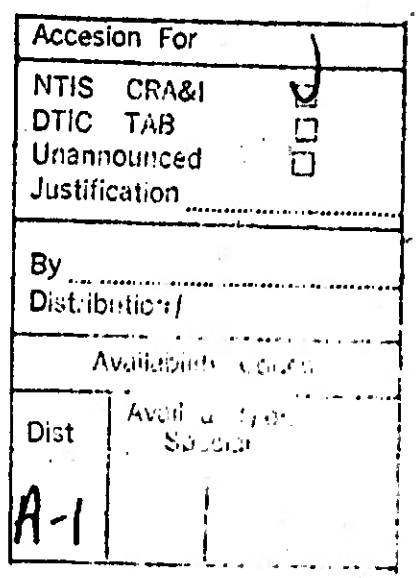
D. W. Laveland
A. W. Biermann

September 30, 1991

U. S. Army Research Office

DAAL03-88-K-0082

Duke University



Approved for Public Release;
Distributian Unlimited

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official department of the Army position, policy, or decision, unless so designated by other documentation.

Contents

1 Note on Appendices

The appendices are the papers and reports prepared under this grant. These are all entries listed in the section "List of Publications and Reports" except for the three Computer Science Department theses listed.

The appendices are physically included in the master copy sent to the project monitor. Other copies (except theses) of the papers are being sent directly to the ARO library.

Note: All papers except the Wilson and Loveland and the Biermann *et al* papers have appeared or are scheduled to appear in journals, conferences with published proceedings or chapters in books. Some of these formats do not provide reprints but are publicly available. We send fifteen copies of each paper in technical report form if reprints are not available. The Wilson and Loveland report (awaiting revision and submission) and the Biermann *et al* paper (submitted) will have fifty copies deposited in the ARO library.

2 Research Summary

2.1 Statement of Problems Studied

This is the final report for ARO Grant DAAL03-88-K-0082. Several different projects were pursued within the context of studying inference and decision mechanisms of Artificial Intelligence (AI). We report on them separately below, although there is interaction on the projects, usually of an informal nature where ideas are discussed and critiqued. Names are associated with those who actively initiate, direct and execute the project in substantial form. In doing so the degree of usefulness of periodic critiquing is lost in the formal documentation of the projects. Other means of cooperation range from jointly being on the doctoral boards of students in each area, to joint papers such as the paper [LN91] by Nadathur and Loveland on logic programming. With this said we present the sections with the name of the principal investigator of that area.

We note that the amount of funds actually provided by this grant is small but stress that the usefulness was disproportionate to the funds. Funds were used in each area to supplement funds from other grants. These funds provided critical student support, faculty support and even transportation of student or faculty to present these results at professional conferences, where useful feedback was obtained as well as broadcasting of the achievements.

In the first section we summarize a revisit to an old technology using modern implementation methods, in the domain of uniform first-order (logic) proof procedures. Uniform proof procedures are truly general purpose in that no specialization to a domain has occurred. Such procedures are particularly valuable in general artificial intelligence projects as general-purpose inference engines. The procedure studied here, Model Elimination (ME), has been used in several language understanding projects, for example. The ME procedure is revisited because of its attractiveness for parallel computer implementation but we learn that the sequential version is surprisingly strong also.

The second section covers the Near-Horn Prolog project, a study (begun before this grant period) to develop the best possible extension of Prolog that addresses the need to handle positive disjunctions. We have reason to believe that we have met our goal, although final evidence in the form of the completed compiler is yet to be gathered. We summarize the reasons for our belief in our success and note the activities done with partial support from this grant. Finally, we note the remaining tasks, which involve application and assessment of the technical success we believe we have realized. As for each of the areas we summarize, the details are in the reports and reprints associated with this final report.

Next we present a summary of the work done on the λProlog extension to Prolog, an extension that incorporates constructs of higher-order logic in the framework of the Prolog setting (i.e., into an extended logic programming setting that preserves the search characteristics and the definite answer characteristics Prolog enjoys). This work also began before this grant was awarded. During this grant period work focussed on a) extending the language, b) implementation issues, and c) applications. To-date we have completed the theoretical

work underlying the promised extension, we have studied several "system" related issues and implementation techniques relevant to the resulting language and have started on an actual implementation based on these studies. We report in detail on the work already completed and indicate the ongoing and remaining work in the appropriate section.

The final section addresses learning and summarizes three studies. The first concerns the foundations of learning; addressed is the question of the capacity to learn and the rate of learning. A tradeoff is established; the larger the space of events that can be learned the slower the learning rate. This is made precise by a "tradeoff" formula. The second study, in the area of connectionist learning theory, involves temporal learning and the question of flexibility for training but with retention capability. The third study examined techniques for synthesizing real-time programs. The latter two studies were Ph.D. theses under Dr. Biermann's direction. Neither student was funded under this grant but both were funded under the previous ARO grant. Both students continued into this grant period using other funding but Dr. Biermann was partially funded by this grant during the period he continued to work with these students.

2.2 Summary of Results

2.2.1 The METEOR prover (D. Loveland)

Historically, depth-first (linear) resolution procedures have not fared well in proving deeper theorems relative to breadth-first resolution provers of various types, primarily because of the search redundancy problem. However, we can now demonstrate that the Model Elimination (ME) procedure, a linear input resolution-like procedure, may be a superior approach for certain types of problems (generally non-Horn problems). The ME theorem provers that provide this demonstration were implemented by Owen Astrachan at Duke, initially as a M.A. thesis project [Ast89] and partially funded by this grant. Development has continued as it has become the basis of work for his Ph.D. dissertation. (The new work on incorporating lemmas and caching within ME, which is an important part of his Ph.D. dissertation, is not addressed here.) The provers, of which there are several variants, are named METEOR provers, METEOR denoting "Model Elimination Theorem Prover with Efficient OR-Parallelism". The name, which now extends to a full family of theorem provers, is based on the first implementation by Owen. This was a parallel implementation on the shared-memory machine at Duke; the BBN Butterfly. The results presented in [AL91] center on the advanced implementations that establish that the ME proof procedure can yield state-of-the-art theorem provers. Studies now underway, including Owen Astrachan's Ph.D. dissertation work, seeks to establish that the ME mechanism can be used in theorem provers that strongly advance the state-of-the-art in first-order proof procedures.

There is a conjunction of reasons why the METEOR provers presently appear so effective. The reasons are: the inherent speed advantage of linear input systems, the sophistication of the WAM architecture in exploiting this advantage, a program written in the language C

using tight coding and effective data structures, the speed of the platforms on which they run, and the successful use of different search strategies.

The METEOR system is really three systems, very highly related; a sequential (single-processor) system, a Butterfly implementation and a system for using a distributted network of (SUN) workstations. We studied the performance of each of these variations, both on problems run on other provers (primarily other ME-based theorem provers, sequential and parallel) and on a difficult graded set of theorems, including the Bledsoe challenge problems given in the Journal of Automated Reasoning in September 1990. No METEOR system can handle all of the challange problems, but to our knowledge only the STR+ve prover of Bledsoe-Hines can solve all of these and in their system special mechanisms handle inequalities. Special handling of designated predicates is not present in the (pure) ME procedure that we use. It is a display of the value of the mechanisms of STR+VE that all these challenge problems are handled by STR+VE; however, we find value in these problems for assessment of the capabilities of METEOR and find it encouraging that no better performance results are known for these problems for resolution theorem provers, other ME provers and similar systems.

One of the most interesting features of the results we obtained is the relative effectiveness of certain depth measures used in the iterative-deepening search. The fact that, for most of the problems we have tried, our best times are often considerably faster than other reported times is certainly partly due to the clock speed of the SUN workstations we have available, also due (we like to think) to the quality of the actual implementation and the effectiveness of the scheduler; but the effectiveness of the depth measure used is key. The proof search uses iterated deepening, the now standard method of realizing completeness in a depth-first search procedure. By iterative deepening we mean expanding the search tree fully to a specified depth, incrementing the depth bound and reexpanding the search tree, and continuing this until a proof is found or the search is terminated. We have investigated two different depth measures and a third which combines the first two may be the most appropiate in many cases. The first depth measure counts the number of inferences in an attempted proof, so every proof attempt gets equal (time) resources. This is a common type of measure in automated theorem provers. The second measure, perhaps unique to ME, counts the size of possible partial models that might invalidate the attempted theorem. If the possible partial models are shown to fail in light of the theorem statement, then that model is discarded, which is clear progress with this set of formulas (clauses) of the theorem and the system continues with this proof direction trying to refute more invalidating possible models. The size of the possible models is usually considerably less than the total number of inferences used (the first depth measure) when progress is being made, but the size grows closer to the total inference count when progress is not made. This measure is often very effective, and was used for the Bledsoe challange problems, but can be ineffective if there are large numbers of possible models of each size. We continue to investigate the usefulness of each depth measure and realize that the results may change in the light of the lemma and caching ideas we are now pursuing.

2.2.2 Near-Horn Prolog (D. Loveland)

The basic goal behind the Near-Horn Prolog project at Duke has been to extend Prolog to disjunctive logic programs (and thus full first-order expressibility) while retaining as much of the clarity and procedural simplicity of Prolog as possible. The approach taken to achieve this goal has been to combine Prolog witb case analysis reasoning. The research work within the project can roughly be divided into three areas: procedure design, semantics, and implementation. Three different variants of Near-Horn Prolog have been devised of which the most recent variant, Inheritance near-Horn Prolog (InH-Prolog), is the variant currently favored. The semantics for tbe near-Horn Prologs, specifically for InH-Prolog, have been investigated, resulting in a case-analysis based fixpoint semantics which mimics the procedural behavior of InH-Prolog. Also, both classical and default negation have been incorporated into the near-Horn Prolog systems. Finally, an interpreter for the original near-Horn Prolog variant was implemented, and a compiler for the InH-Prolog variant is currently nearing completion.

As stated above, the approach to the design of the near-Horn procedures was to combine Prolog with case analysis reasoning. In [Lov87, Lov91], we developed the procedure now known as **Unit near-Horn Prolog (UnH-Prolog)** (originally called "Progressive nH-Prolog"), as well as the conceptually simpler but incomplete variant **Naive near-Horn Prolog (NnH-Prolog)**. Following these, we (Loveland and Reed [LR89]) developed the **Inheritance near-Horn Prolog (InH-Prolog)** variant that is simpler and sometimes permits shorter proofs, but may have a lower lips rate than the earlier nH-Prologs. (The /it lips rate refers to logical inferences per second.) These near-Horn Prolog procedures have several features which make them desirable when compared to existing procedures such as Model Elimination [Lov78] and SLI-resolution [MZ82]. (Note: the SLI-resolution procedure is based on the Model Elimination *(ME)* procedure that was developed two decades ago by us (Loveland). These procedures are actively being studied by others as extensions of Prolog, including for disjunctive logic programming. The near-Horn procedures were developed by us to specifically address some of the shortcomings of ME *when perceived as a logic programming language.* The ME procedure is a very useful procedure in different contexts such as automated theorem proving or simply as an inference engine for artificial intelligence devices.)

We now address the major advantages of nH-Prolog systems over ME-style systems when intended as logic programming languages (systems). First, the nH-Prolog family implements a positive implication logic, as Prolog does, meaning that clauses are in implication form and use only positive literals (atoms). This allows the same syntax for negation-as-failure as used by Prolog; the negative literal in the clause body. (Negation-as-failure is a means of implementing negation that marks the negative statement as true when the positive statement fails to be shown in a finite length searcb. It is used strongly by the database community on the basis that all the true significant facts are listed, hence retreivable.) The positive implication nature of the nH-Prologs also limits the number of contrapositives of program clauses

which must be considered, and preserves the direction of information flow (goal-to-body) in these contrapositives. Second, the procedures are full first-order proof procedures (as are the above alternatives) providing the expressive power of the full logic, including classical negation. However, bere the classical negation is encoded within the positive implication logic, which allows the visual separation of classical negation from negation-as-failure. Third, the lips rate is bigher than for ME or SLI-resolution. For the latter procedures a basic operation requires addressing the list of ancestor literals, which grows with proof depth. However, for NnH-Prolog or UnH-Prolog only one added atom must be consulted at most, and for InH-Prolog only as many atoms as the nested case-analysis needs locally. In practice this seems rarely to exceed two atoms. This brings the fourth point, that the performance degrades "gracefully" as the program becomes more non-Horn, rather than growing with proof depth as soon as one non-Horn clause is used. Fifth, and finally, the nH-Prolog procedures possess both a declarative and a (local) procedural semantics. The procedural reading that can be given the program is not shared by ME procedures. The procedural reading for nH-Prolog programs is very close to that of Prolog; the addition is that a non-Horn clause is read as a multientry procedure, where the atoms in the head that are not chosen are ignored (for the duration of the case under consideration). These characteristics of nH-Prolog procedures convince us that we have found the correct formulation for a first-order proof procedure if the goal is to preserve as many of the properties associated with Horn-clause logic programming as possible. The price paid is relatively poor performance if many non-Horn clauses exist in the program, an event we think will be rare in an application done witbin the logic programming paradigm.

The most recent work on the nH-Prolog project has involved the fixpoint characterization of InH-Prolog (and the UnH-Prolog implicitly; an understood modification yields similar results for UnH-Prolog.) This work is reprted in [RLS91] and is developed in full detail in [Ree91]. This gives a way of understanding how a negation-as-failure concept of negation fits with the nH-Prolog procedures and provides some other insights into the procedures. Also, an important task has been to build a "protocompiler"for InH-Prolog. The "proto-" denotes that we are demonstrating certain design and performance characteristics and thus are not developing the full-efficiency compiler. In particular the protocompiler is implemented in Quintus Prolog, an expanded Prolog allowing access to C-programs, wbich are system-level-coded programs used for critical parts of the protocompiler. The interpreter is documented in [SL88]; the protocompiler is not yet completed nor yet documented in the literature. (It is nearing completion.)

Tbere is considerable activity in the area of disjunctive logic programming at present, demonstrated by the workshop on disjunctive kogic programming meeting in October, 1991, wbere 10 papers are being presented. However, there is a real question of the area of application for this technology. It is clear that there should be such because we have in theory extended logic programming to all of first-order logic, the classic representation language within the artificial intelligence world, for example. We do have many examples for illustration, some of which have appeared in our papers. Also, we have given considerable thought

to the quesstion of large-scale examples and areas of primary application. We presently feel that the (only) natural area of application is the planning area, because there the reliance on a single canonical (minimal) model, which is the key to Horn-clause logic which Prolog implements, is no longer valid. Planning involves multiple models with the "correct" model to be determined by time or refinement of knowledge. To undertake planning within the contex of logic programming will take time to develop but we have just learned of a real application of such a technique to a planning study involving the Great Barrier Reef in Australia. Such use will multiply in the future, we believe.

2.2.3 Lambda Prolog (G. Nadathur)

One component of the research originally proposed involved developing the higher-order logic programming language called λProlog. Work had previously been conducted on this language in collaboration with Dale Miller from the University of Pennsylvania, and the status of this work prior to the grant period was the following: The theoretical underpinnings of what is now a subset of λProlog had been examined [Nad87] and an experimental implementation for the resulting language had been undertaken. This implementation had been used to demonstrate important applications for λProlog as a programming language for implementing derivation systems [MN87, Nad87]. Experiments with the implementation had, however, indicated the necessity for including certain new primitives in the language in order to realize its full potential in the role of a metalanguage. Motivated by this experience, we had proposed (a) to undertake theoretical work to extend the language so as to include the desired programming features, (b) to investigate issues pertaining to implementing the resulting language efficiently and (c) to examine the applications of the language in a more complete fashion. Progress has been made on the first two aspects under this grant and, while we have not had the resources to devote to the last aspect ourselves, several of our colleagues have explored the application realm with much success (e.g. see [FM88, HM90, Pfe88]). In the paragraphs below, we explain in greater detail the specific work that has been supported in part by the grant.

The original basis of λProlog was the logical theory of higher-order Horn clauses [NM90]. The language that resulted from using these formulas turned our to be very useful from the perspective of implementing derivation systems for two reasons. First, the underlying programming unit was a Horn clause and as such provided a very natural means for implementing inference rules that characterized typical derivation systems. Second, the data structures that were provided by virtue of the "higher-orderness" of the language were very well suited for representing syntactically complex objects in a natural manner; thus derivation systems that manipulated extremely complicated entities could be easily implemented within the framework. However, there was a lacuna in the language given its application realm. In constructing subparts of a typical derivation, there is often a need to assume the existence of new entities as well as the truth of new facts; such an ability is needed, for instance, in constructing a proof of a universal or an implication statement in first-order logic.

The Horn clause fragment unfortunately does not provide primitives for readily realizing these "assumption" capabilities.

The first task that we undertook, then, was to add primitives for assuming new objects and new facts to the Horn clause framework while preserving properties that made the fragment useful for logic programming. At the very outset, this required understanding what the relationship between logic programming and Horn clauses is and describing ways in which Horn clauses may be extended while preserving this relationship. Our work on this aspect resulted in the notion of *uniform proofs* that is described in [MNPS]. This notion was used to outline a criterion for determining whether or not a given logical language was a suitable basis for logic programming. On the constructive side, this criterion was exploited in providing a very rich extension within intuitionistic logic to Horn clauses: the higher-order *hereditary Harrop formulas* described in [MNPS]. This extension to Horn clause logic provides a means not only for realizing the assumption capabilities mentioned above, but also for incorporating several abstraction mechanisms (such as modules) within logic programming. Both the notion of uniform proofs and our specific extension to Horn clause logic have turned out to be fairly significant results within logic programming: the first has been used in describing other extensions and the second has stimulated attempts to realize new programming language features using aspects of our extension.

The second task that we undertook was that of providing an efficient implementation for the logic of higher-order hereditary Harrop formulas. The similarities between this logic and that of Horn clauses, argued for the use of the technology of the "Warren Abstract Machine" (WAM) in implementing this language as well. However, a means to deal with significant new problems have to be devised before a satisfactory adaptation is obtained. In particular, the following facets of the extended language have to be dealt with:

(1) the use of "higher-order" terms and the need to perform function evaluation on these terms,

(2) the embedding of higher-order unification with its branching characteristic within the normal Prolog computation regime,

(3) the use of a typing regime that, within logic programming, leads to a new set of computations to be performed at run-time, and

(4) the presence of two new "search" primitives that provide the assumption capabilities discussed above.

We have made much progress under the grant in understanding the true nature of the implementation problem posed by each of these features and in detailing methods for dealing with these problems. In joint work with Bharat Jayaraman from SUNY Buffalo, we have explored the necessary additions to the WAM in order to deal with higher-order unification and the new search primitives [JN91, NJ89]. We have, in fact, proposed implementation

schemes for languages that incorporate each of these features and these schemes have several pleasing properties: the resulting machinery appears to be conservative over the Prolog subset of the language, the additions required are relatively well-controlled and the overall framework supports a fair degree of compilation. We have understood clearly the true nature of types in our language and also the implementation problems they raise [NP91] and, in joint work with two graduate students, have developed a scheme for dealing with them. The two main aspects of this scheme is that it permits much runtime type checking to be eliminated by compile time analysis and it generates compiled code to deal with the rest. In conjunction with the first aspect listed above, we are exploring representations of higher-order terms that are suitable in our context with Debra Wilson. We have succeeded in describing a representation that appears satisfactory for several purposes [NW90] and are studying its properties in ongoing work that has been partially supported by this grant. Finally, we have brought these various components of our work together in the design of an abstract machine for the logical language underlying λProlog.

We had originally hoped to bring our implementation ideas to fruition in an actual compiler for λProlog within the grant period. This goal, in retrospect, was extremely ambitious and has not been completely achieved. We are however in the the process of developing such an implementation. We hope to have an early version of this implementation available for release to a "friendly" user base in the summer of 1992 with the purpose of getting a feedback on the techniques used and of determining improvements to be made.

2.2.4 Learning (A. Biermann)

Associated with each learning system is a class of learnable behaviors. If the target behavior to be acquired is in the learnable class, it will be learned perfectly. If it is outside that class, the machine will only be able to acquire a behavior that approximates the target and it will always make errors. It is desirable for a learning machine to have a large learnable class to maximize the chances of acquiring the unknown behavior and to minimize the expected error when only an approximation is possible. However, it is also desirable to have a small learnable class so that learning can be achieved rapidly. Thus the design of learning machines involves selecting a position on the spectrum: minimum error and slow learning time versus larger error and faster learning time.

We study the class learning machines that receive a binary input vector and compute a single binary value output and where learning is done by generalizing on examples of target input-output pairs. We derive a relationship that makes explicit the tradeoff described above. Specifically, we show that

$$(\log_2(L_p))/2^p >= 1 - H(\beta)$$

where p is the number of binary inputs, L_p is the number of learnable behaviors, $\log_2(L_p)$ is the minimum number of examples required to learn, and $H(\beta)$ is a function of the maximum allowed error. If the machine is required to learn with great accuracy, then $H(\beta)$ will be very

small, meaning that the number of examples required to learn will be nearly 2^p. That is, it is necessary to observe almost every one of the possible input-output behaviors in order to learn if the allowed error rate is small. If the allowed error rate is large, then $H(\beta)$ will be larger, meaning that learning can be done with less information.

We define a class of learning machines to be "realization sparse" if the left side of the above relationship approaches zero as p becomes large. Such a class has the property that it can learn quickly but, for large p, it learns a disappearingly small class of behaviors. We show that common classes of learning machines such as the signature tables, the linear learning machines, and the conjunctive normal form learning machines are realization sparse in this sense.

In a second study, we examine the classic "stability-plasticity" dilemma of connectionist learning theory. The problem here is that a system must be plastic enough to adapt to new incoming sequences and stable enough to hold them once they are learned even though many other patterns need to be learned. We have developed a neural network to solve this problem. The network is composed of two fields F^1 and F^2 where F^1 converts temporal sequences to special patterns. F^2 is an on-center off-surround network that obeys winner-take-all dynamics and is similar to a Grossberg-Carpenter architecture . At F^2, new classifications can form without degrading previous classifications. The technique used is to make F^2 a nonhomogeneous field. Nodes learn different output characteristics so that different nodes can respond preferentially to different size patterns. Nonuniform inhibitory connections are learned at F^2 to allow nodes to compete only with other nodes coding similar patterns. Extensive simulations demonstrate the viability of the approach. (This project was done by Albert Nigrin as a Ph.D. dissertation.)

In a third study, we examine techniques for synthesizing real time programs from examples of their behaviors. A real time program is defined here to be a program that reads a finite sequence of symbols and performs one computational step after each such symbol. The methodology requires a finite subset of target behaviors and creates the smallest finite graph B that can account for the behaviors. Then the graph is factored into two graphs, C and D, which represent respectively, the control and data structures for the target program. We show that by the construction, B=C.D where . is a graph product defined in a straightforward way. The decomposition methodology uses the technique of Hartmanis-Stearns and has been demonstrated to be capable of creating a variety of interesting real time programs. (This was done by Amr Fahmy as a Ph.D dissertation.)

3 Publications, Technical Reports and Ph.D. Theses Partially Funded by this Grant

[1] O.L. Astrachan. METEOR: Model Elimination theorem proving for efficient OR-parallelism. M.S. Thesis, C.S. Dept., Duke Univ., Apr. 1989.

[2] O.L. Astrachan and D.W. Loveland. METEORs; high performance theorem provers for Model Elimination. In R.S. Boyer, editor, *Automoted Reosoning: Essoys in Honor of Woody Bledsoe.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991.

[3] Biermann, A.W., K.C. Gilbert, A. Fahmy, B. Koster. On the errors that learning machines will make. Submitted to *Artificial Intelligence*. 57 pp.

[4] Fahmy, A.F. Synthesis of real-time programs. Ph.D. dissertation, Department of Computer Science, Duke University. 1988. v + 118 pp.

[5] Jayaraman, B. and G. Nadathur. Implementation techniques for scoping constructs in logic programming. Proceedings of the Eighth International Conference on Logic Programming, MIT Press, 1991, 871 - 889.

[6] Loveland, D.W. Near-Horn Prolog and beyond. *Journol of Automoted Reosoning*, 7, 1991, 1 - 26.

[7] Loveland, D.W. and D.W. Reed. A Near-Horn Prolog for compilation. To appear in *Computationol Logic: Essoys in Honor of Alon Robinson*, MIT Press.

[8] Miller, D., G. Nadathur, F. Pfenning, A. Scedrov. Uniform Proofs as a foundation for logic programming. *Annols of Pure ond Appl. Logic*, 51 (1991) 125 - 157.

[9] Nadathur, G. and B. Jayaraman. Towards a WAM Model for λProlog. *Proceedings of the North Americon Conference on Logic Progromming*, MIT Press, 1989, 1180 - 1198.

[10] Nigrin, A. The stable learning of temporal patterns with an adaptive resonance circuit. Ph.D. dissertation, Department of Computer Science, Duke University, 1990, vii + 338 pp.

[11] Reed D.W. and D.W. Loveland. A comparison of three Prolog extensions. To appear in *Journol of Logic Programming.*

[12] Reed, D.W., D.W. Loveland and B.T. Smith. An alternative characterization of disjunctive logic programs. *Proceedings of the 1991 Internotionol Symposium on Logic Programming*, San Diego, Oct. 1991.

[13] Wilson, D.S. and D.W. Loveland. Incorporating relevance testing in SATCHMO. Computer Science Technical Report CS-1989-24, Duke University, Nov. 1989, 15 pp.

4 Participating Scientific Personnel

Student	*Degree Earned*
Owen L. Astrachan	M.A. (Ph.D. in progress)
David W. Reed	(Ph.D. nearly completed)
Debra Sue Wilson	(Ph.D. in progress)

Senior Investigators

Alan W. Biermann
Donald W. Loveland
Gopalan Nadathur

References

[Ast89] O.L. Astrachan. METEOR: Model Elimination theorem proving for efficient OR-parallelism. M.S. Thesis, C.S. Dept., Duke Univ., Apr. 1989.

[AL91] O.L. Astrachan and D.W. Loveland. METEORs; high performance theorem provers for Model Elimination. In R.S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991.

[FM88] A. Felty and D. Miller. Specifying Theorem Provers in a Higher-Order Logic Programming Language. *Proceedings of CADE-9*, pages 61 – 80, Springer-Verlag. 1988.

[HM90] J. Hannan and D. Miller. From Operational Semantics to Abstract Machines: Preliminary Results. *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 323 – 333, ACM Press, 1990.

[JN91] B. Jayaraman and G. Nadathur. Implementation Techniques for Scoping Constructs in Logic Programming. *Proceedings of the Eighth International Conference on Logic Programming*, MIT Press, 1991.

[Lov78] D.W. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, Amsterdam, 1978.

[Lov87] D.W. Loveland. Near-Horn Prolog. In J. Lassez, editor, *Logic Programming: Proc. of the Fourth Int'l Conf.*, pages 456–469. MIT Press, 1987.

[Lov91] D.W. Loveland. Near-Horn Prolog and beyond. *J. Automated Reasoning*, 7:1–26, 1991.

[LN91] D.W. Loveland and G. Nadathur. Proof Procedures for Logic Programming. To appear in *Handbook of Logic in Artificial Intelligence and Logic Programming*, Oxford University Press.

[LR89] D.W. Loveland and D.W. Reed. A near-Horn Prolog for compilation. Technical Report CS-1989-14, Duke University, 1989. To appear in *Computational Logic: Essays in Honor of Alan Robinson.*

[MN87] D. Miller and G. Nadathur. A Logic Programming Approach to Manipulating Formulas and Programs. *Proceedings of the IEEE Fourth Symposium on Logic Programming*, pages 379 – 388, IEEE Press, 1987.

[MNPS] D. Miller, G. Nadathur, F. Pfenning and A. Scedrov. Uniform Proofs as a Foundation for Logic Programming. *Annals of Pure and Appl. Logic*, 51 (1991) 125 – 157.

[MZ82] J. Minker and G. Zanon. An extension to linear resolution with selection function. *Information Processing Letters*, 14(3):191–194, 1982.

[Nad87] G. Nadathur. *A Higher-Order Logic as the Basis for Logic Programming.* PhD thesis, University of Pennsylvania, May 1987.

[NJ89] G. Nadathur and B. Jayaraman. Towards a WAM Model for λProlog. *Proceedings of the North American Conference on Logic Programming*, pages 1180 – 1198, MIT Press, 1989.

[NM90] G. Nadathur and D. Miller. Higher-Order Horn Clauses. *J. ACM*, Vol 37, No 4, pages 777 – 814, 1990.

[NP91] G. Nadathur and F. Pfenning. The Type System of a Higher-Order Logic Programming Language. To appear in *Types in Logic Programming.* MIT Press, 1991.

[NW90] G. Nadathur and D. S. Wilson. A Pepresentation of Lambda Terms Suitable for Pperations on their Intensions. *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 341 – 348, ACM Press, 1990.

[Pfe88] F. Pfenning. Partial Polymorphic Type Inference and Higher-Order Unification. *Proceedings of the ACM Lisp and Functional Programming Conference*, pages 153 – 164, ACM Press, 1988.

[Ree91] D.W. Reed. *A Case-analysis Approach to Disjunctive Logic Programming.* PhD thesis, Duke University, Decenber 1991.

[RLS91] D.W. Reed, D.W. Loveland, and B.T. Smith. An alternative characterization of disjunctive logic programs. Technical Report CS-1991-27, Duke University, 1991. To appear in *Proc. 1991 Int'l Logic Programming Symp.*

[SL88] B.T. Smith and D.W. Loveland. A simple near-Horn Prolog interpreter. In Kowalski and Bowen, editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 794–809. MIT Press, 1988.